第四次上机报告(lab_05)

运行示例代码

将完整示例代码写入对应脚本，代码如下

vrmp.h

```
#include <malloc.h>
#include <iomanip>
#include <iostream>

using namespace std;

class Replace {
   public:
    Replace();
    ~Replace();
    void InitSpace(char* MethodName);  // 初始化页号记录
    void Report(void);                 // 报告算法执行情况
    void Fifo(void);                   // 先进先出算法
    void Lru(void);                    // 最近最旧未用算法
    void Clock(void);                  // 时钟(二次机会)置换算法
    void Eclock(void);                 // 增强二次机会置换算法
    void Lfu(void);                    // 最不经常使用置换算法
    void Mfu(void);                    // 最经常使用置换算法
   private:
    int* ReferencePage;  // 存放要访问到的页号
    int* EliminatePage;  // 存放淘汰页号
    int* PageFrames;     // 存放当前正在实存中的页号
    int PageNumber;      // 访问页数
    int FrameNumber;     // 实存帧数
    int FaultNumber;     // 失败页数
};
```

vmrp.cc

```
#include "vmrp.h"

Replace::Replace() {
    int i;
    // 设定总得访问页数,并分配相应的引用页号和淘汰页号记录数组空间
    cout << "Please input page numbers:";
    cin >> PageNumber;
    ReferencePage = new int[sizeof(int) * PageNumber];
    EliminatePage = new int[sizeof(int) * PageNumber];
    // 输入引用页号序列(页面走向),初始化引用页数组
    cout << "Please input reference page string:";
    for (i = 0; i < PageNumber; i++)
        cin >> ReferencePage[i];  // 引用页暂存引用数组
    // 设定内存实页数(帧数),并分配相应的实页号记录数组空间(页号栈)
    cout << "Please input page frames:";
```

```
    cin >> FrameNumber;
    PageFrames = new int[sizeof(int) * FrameNumber];
}

Replace::~Replace() {}

void Replace::InitSpace(char* MethodName) {
    int i;
    cout << endl << MethodName << endl;
    FaultNumber = 0;
    // 引用还未开始,-1表示无引用页
    for (i = 0; i < PageNumber; i++)
        EliminatePage[i] = -1;
    for (i = 0; i < FrameNumber; i++)
        PageFrames[i] = -1;
}

// 分析统计选择的算法对于当前输入的页面走向的性能
void Replace::Report(void) {
    // 报告淘汰页顺序
    cout << endl << "Eliminate page:";
    for (int i = 0; EliminatePage[i] != -1; i++)
        cout << EliminatePage[i] << " ";
    // 报告缺页数和缺页率
    cout << endl << "Number of page faults=" << FaultNumber << endl;
    cout << setw(6) << setprecision(3);
    cout << "Rate of page faults=" << 100 * (float)FaultNumber /
(float)PageNumber << "%" << endl;
}

// 最近最旧未用置换算法
void Replace::Lru(void) {
    int i, j, k, l, next;
    InitSpace("LRU");
    // 循环装入引用页
    for (k = 0, l = 0; k < PageNumber; k++) {
        next = ReferencePage[k];
        // 检测引用页当前是否已在实存
        for (i = 0; i < FrameNumber; i++) {
            if (next == PageFrames[i]) {
                // 引用页已在实存将其调整到页记录栈顶
                next = PageFrames[i];
                for (j = i; j > 0; j--)
                    PageFrames[j] = PageFrames[j - 1];
                PageFrames[0] = next;
                break;
            }
        }
        if (PageFrames[0] == next) {
            // 如果引用页已放栈顶，则为不缺页，报告当前内存页号
            for (j = 0; j < FrameNumber; j++)
                if (PageFrames[j] >= 0)
                    cout << PageFrames[j] << " ";
            cout << endl;
            continue;  // 继续装入下一页
        } else {
            // 如果引用页还未放栈顶,则为缺页,缺页数加 1
            FaultNumber++;
```

```
            // 栈底页号记入淘汰页数组中
            EliminatePage[l] = PageFrames[FrameNumber - 1];
            // 向下压栈
            for (j = FrameNumber - 1; j > 0; j--)
                PageFrames[j] = PageFrames[j - 1];
            PageFrames[0] = next;  // 引用页放栈顶
            // 报告当前实存中页号
            for (j = 0; j < FrameNumber; j++)
                if (PageFrames[j] >= 0)
                    cout << PageFrames[j] << " ";
            // 报告当前淘汰的页号
            if (EliminatePage[l] >= 0)
                cout << "->" << EliminatePage[l++] << endl;
            else
                cout << endl;
        }
    }
    Report();
}

// 先进先出置换算法
void Replace::Fifo(void) {
    int i, j, k, l, next;
    InitSpace("FIFO");
    // 循环装入引用页
    for (k = 0, j = l = 0; k < PageNumber; k++) {
        next = ReferencePage[k];
        // 如果引用页已在实存中，报告实存页号
        for (i = 0; i < FrameNumber; i++)
            if (next == PageFrames[i])
                break;
        if (i < FrameNumber) {
            for (i = 0; i < FrameNumber; i++)
                cout << PageFrames[i] << " ";
            cout << endl;
            continue;  // 继续引用下一页
        }
        // 引用页不在实存中，缺页数加1
        FaultNumber++;
        EliminatePage[l] = PageFrames[j];  // 最先入页号记入淘汰页数组
        PageFrames[j] = next;              // 引用页号放最先入页号处
        j = (j + 1) % FrameNumber;         // 最先入页号循环下移
        // 报告当前实存页号和淘汰页号
        for (i = 0; i < FrameNumber; i++)
            if (PageFrames[i] >= 0)
                cout << PageFrames[i] << " ";
        if (EliminatePage[l] >= 0)
            cout << "->" << EliminatePage[l++] << endl;
        else
            cout << endl;
    }
    Report();
}

// 未实现的其他页置换算法入口
void Replace::Clock(void) {}
void Replace::Eclock(void) {}
void Replace::Lfu(void) {}
```

```
void Replace::Mfu(void) {}

int main(int argc, char* argv[]) {
    Replace* vmpr = new Replace();
    vmpr->Lru();
    vmpr->Fifo();
    return 0;
}
```

编译运行后，得到结果如下：

```
kpmark@LAPTOP-AITJPLGC:/mnt/d/os-homework/os_lab_5$ ./vmrp
Please input page numbers:12
Please input reference page string:1 2 3 4 1 2 5 1 2 3 4 5
Please input page frames:3

LRU
1
2 1
3 2 1
4 3 2 ->1
1 4 3 ->2
2 1 4 ->3
5 2 1 ->4
1 5 2
2 1 5
3 2 1 ->5
4 3 2 ->1
5 4 3 ->2

Eliminate page:1 2 3 4 5 1 2
Number of page faults=10
Rate of page faults=83.3%

FIFO
1
1 2
1 2 3
4 2 3 ->1
4 1 3 ->2
4 1 2 ->3
5 1 2 ->4
5 1 2
5 1 2
5 3 2 ->1
5 3 4 ->2
5 3 4

Eliminate page:1 2 3 4 1 2
Number of page faults=9
Rate of page faults=75%
```

补全缺失算法

有如下算法需要补全

- void Replace::Clock(void) {}
- void Replace::Eclock(void) {}
- void Replace::Lfu(void) {}
- void Replace::Mfu(void) {}

补全代码如下：

```
// CLOCK 算法实现
void Replace::Clock(void) {
    int i, k, l, next;
    vector<bool> referenceBit(FrameNumber, false);
    InitSpace("CLOCK");

    for (k = 0, l = 0; k < PageNumber; k++) {
        next = ReferencePage[k];

        // 检查页面是否已在内存中
        for (i = 0; i < FrameNumber; i++) {
            if (next == PageFrames[i]) {
                referenceBit[i] = true;  // 设置引用位
                break;
            }
        }

        if (i < FrameNumber) {
            // 页面命中
            for (i = 0; i < FrameNumber; i++)
                if (PageFrames[i] >= 0)
                    cout << PageFrames[i] << " ";
            cout << endl;
        } else {
            // 页面缺失
            FaultNumber++;

            // 寻找替换页面
            while (true) {
                if (PageFrames[clockPointer] == -1) {
                    // 空闲帧
                    PageFrames[clockPointer] = next;
                    referenceBit[clockPointer] = true;
                    clockPointer = (clockPointer + 1) % FrameNumber;
                    break;
                } else if (!referenceBit[clockPointer]) {
                    // 找到替换页面
                    EliminatePage[l] = PageFrames[clockPointer];
                    PageFrames[clockPointer] = next;
                    referenceBit[clockPointer] = true;
                    clockPointer = (clockPointer + 1) % FrameNumber;
                    break;
                } else {
                    // 给第二次机会
                    referenceBit[clockPointer] = false;
                    clockPointer = (clockPointer + 1) % FrameNumber;
                }
            }

            for (i = 0; i < FrameNumber; i++)
                if (PageFrames[i] >= 0)
                    cout << PageFrames[i] << " ";

            if (EliminatePage[l] >= 0)
                cout << "->" << EliminatePage[l++] << endl;
```

```
            else
                cout << endl;
        }
    }
    Report();
}

// eclock
void Replace::Eclock(void) {
    int k, l, next;
    InitSpace("Enhanced CLOCK");

    for (k = 0, l = 0; k < PageNumber; k++) {
        next = ReferencePage[k];

        // 检查页面是否已在内存中
        int found = -1;
        for (int i = 0; i < FrameNumber; i++) {
            if (EnhancedFrames[i].pageNum == next) {
                found = i;
                EnhancedFrames[i].referenced = true;
                // 模拟修改位
                if (rand() % 3 == 0) {
                    EnhancedFrames[i].modified = true;
                    cout << " modified" << endl;
                }
                break;
            }
        }

        if (found != -1) {
            // 页面命中
            for (int i = 0; i < FrameNumber; i++)
                if (EnhancedFrames[i].pageNum >= 0)
                    cout << EnhancedFrames[i].pageNum << " ";
            cout << endl;
        } else {
            // 页面缺失
            FaultNumber++;

            // 寻找替换页面 - 增强二次机会算法
            int victim = -1;
            int startPointer = clockPointer;

            // 第一轮：寻找 (0,0) - 未引用且未修改
            do {
                if (EnhancedFrames[clockPointer].pageNum == -1) {  // 空闲帧
                    victim = clockPointer;
                    break;
                }
                if (!EnhancedFrames[clockPointer].referenced &&
!EnhancedFrames[clockPointer].modified) {
                    victim = clockPointer;
                    break;
                }
                clockPointer = (clockPointer + 1) % FrameNumber;
            } while (clockPointer != startPointer);
```

```
            // 第二轮：寻找 (0,1) - 未引用但已修改，同时清除引用位
            if (victim == -1) {
                do {
                    if (!EnhancedFrames[clockPointer].referenced &&
EnhancedFrames[clockPointer].modified) {
                        victim = clockPointer;
                        break;
                    }
                    if (EnhancedFrames[clockPointer].referenced) {
                        EnhancedFrames[clockPointer].referenced = false;
                    }
                    clockPointer = (clockPointer + 1) % FrameNumber;
                } while (clockPointer != startPointer);
            }

            // 第三轮：寻找 (0,0) - 在清除引用位后
            if (victim == -1) {
                do {
                    if (!EnhancedFrames[clockPointer].referenced &&
!EnhancedFrames[clockPointer].modified) {
                        victim = clockPointer;
                        break;
                    }
                    clockPointer = (clockPointer + 1) % FrameNumber;
                } while (clockPointer != startPointer);
            }

            // 第四轮：寻找 (0,1)
            if (victim == -1) {
                do {
                    if (!EnhancedFrames[clockPointer].referenced) {
                        victim = clockPointer;
                        break;
                    }
                    clockPointer = (clockPointer + 1) % FrameNumber;
                } while (clockPointer != startPointer);
            }

            // 执行替换
            if (EnhancedFrames[victim].pageNum >= 0) {
                EliminatePage[l++] = EnhancedFrames[victim].pageNum;
            }

            EnhancedFrames[victim].pageNum = next;
            EnhancedFrames[victim].referenced = true;
            EnhancedFrames[victim].modified = (rand() % 3 == 0);

            clockPointer = (victim + 1) % FrameNumber;

            for (int i = 0; i < FrameNumber; i++)
                if (EnhancedFrames[i].pageNum >= 0)
                    cout << EnhancedFrames[i].pageNum << " ";

            if (l > 0 && EliminatePage[l - 1] >= 0)
                cout << "->" << EliminatePage[l - 1] << endl;
            else
                cout << endl;
        }
```

```
    }
    Report();
}

// LFU 算法实现
void Replace::Lfu(void) {
    int i, k, l, next, victim;
    InitSpace("LFU");

    for (k = 0, l = 0; k < PageNumber; k++) {
        next = ReferencePage[k];

        // 检查页面是否已在内存中
        for (i = 0; i < FrameNumber; i++) {
            if (next == PageFrames[i]) {
                PageFrequency[i]++;
                break;
            }
        }

        if (i < FrameNumber) {
            // 页面命中
            for (i = 0; i < FrameNumber; i++)
                if (PageFrames[i] >= 0)
                    cout << PageFrames[i] << " ";
            cout << endl;
        } else {
            // 页面缺失
            FaultNumber++;

            // 寻找空闲帧或最少使用的页面
            victim = 0;
            for (i = 0; i < FrameNumber; i++) {
                if (PageFrames[i] == -1) {
                    victim = i;
                    break;
                }
                if (PageFrequency[i] < PageFrequency[victim]) {
                    victim = i;
                }
            }

            if (PageFrames[victim] >= 0) {
                EliminatePage[l++] = PageFrames[victim];
            }

            PageFrames[victim] = next;
            PageFrequency[victim] = 1;

            for (i = 0; i < FrameNumber; i++)
                if (PageFrames[i] >= 0)
                    cout << PageFrames[i] << " ";

            if (l > 0 && EliminatePage[l - 1] >= 0)
                cout << "->" << EliminatePage[l - 1] << endl;
            else
                cout << endl;
        }
```

```
    }
    Report();
}

void Replace::Mfu(void) {
    int i, k, l, next, victim;
    InitSpace("MFU");

    for (k = 0, l = 0; k < PageNumber; k++) {
        next = ReferencePage[k];

        // 检查页面是否已在内存中
        for (i = 0; i < FrameNumber; i++) {
            if (next == PageFrames[i]) {
                PageFrequency[i]++;
                break;
            }
        }

        if (i < FrameNumber) {
            // 页面命中
            for (i = 0; i < FrameNumber; i++)
                if (PageFrames[i] >= 0)
                    cout << PageFrames[i] << " ";
            cout << endl;
        } else {
            // 页面缺失
            FaultNumber++;

            // 寻找空闲帧或最常使用的页面
            victim = 0;
            for (i = 0; i < FrameNumber; i++) {
                if (PageFrames[i] == -1) {
                    victim = i;
                    break;
                }
                if (PageFrequency[i] > PageFrequency[victim]) {
                    victim = i;
                }
            }

            if (PageFrames[victim] >= 0) {
                EliminatePage[l++] = PageFrames[victim];
            }

            PageFrames[victim] = next;
            PageFrequency[victim] = 1;

            for (i = 0; i < FrameNumber; i++)
                if (PageFrames[i] >= 0)
                    cout << PageFrames[i] << " ";

            if (l > 0 && EliminatePage[l - 1] >= 0)
                cout << "->" << EliminatePage[l - 1] << endl;
            else
                cout << endl;
        }
    }
```

```
    Report();
}
```

```
CLOCK
1
1 3
1 3 2
1 3 2
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5 4
1 3 2 5 4
1 3 2 5 4

Eliminate page:
Number of page faults=5
Rate of page faults=41.7%

Enhanced CLOCK
1
1 3
1 3 2
1 3 2
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5
 modified
1 3 2 5
1 3 2 5 4
1 3 2 5 4
 modified
1 3 2 5 4

Eliminate page:
Number of page faults=5
Rate of page faults=41.7%
```

```
LFU
1
1 3
1 3 2
1 3 2
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5 4
1 3 2 5 4
1 3 2 5 4

Eliminate page:
Number of page faults=5
Rate of page faults=41.7%

MFU
1
1 3
1 3 2
1 3 2
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5
1 3 2 5 4
1 3 2 5 4
1 3 2 5 4

Eliminate page:
Number of page faults=5
Rate of page faults=41.7%
```

发现的现象

Belady异常现象

在某些情况下，增加物理帧数反而会增加缺页次数，特别是在FIFO算法中较为明显。

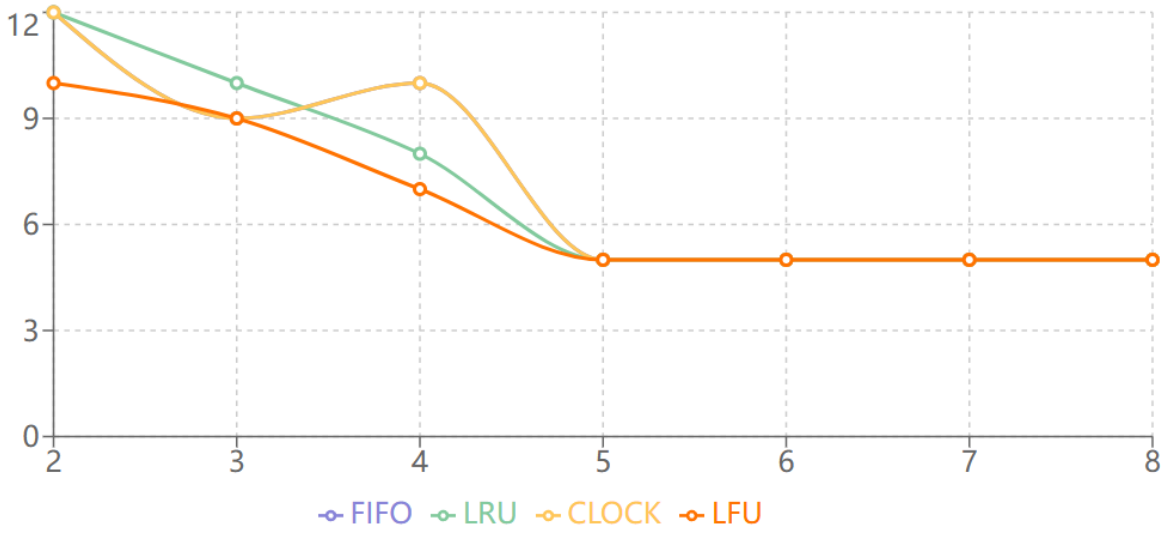
算法性能差异

- **FIFO**: 实现简单但性能不稳定，容易出现Belady异常
- **LRU**: 性能较好但实现复杂度高
- **CLOCK**: 性能接近LRU但实现更简单
- **Enhanced CLOCK**: 考虑修改位，减少不必要的页面写回
- **LFU/MFU**: 在特定访问模式下表现优异

局部性原理验证

具有良好局部性的引用串在所有算法中都表现更好，缺页率显著降低。

各算法适应性分析



FIFO算法

适用场景:

- 实现简单性要求高的系统
- 内存访问模式相对均匀的应用

不适用场景:

- 具有强局部性的应用
- 对性能要求严格的系统

LRU算法

适用场景:

- 具有时间局部性的应用
- 对缺页率要求严格的系统
- 内存充足的环境

不适用场景:

- 硬件资源受限的系统
- 实时性要求极高的应用

CLOCK算法

适用场景:

- 需要在性能和实现复杂度间平衡的系统
- 大多数通用操作系统
- 中等规模的应用系统

Enhanced CLOCK算法

适用场景:

- I/O密集型应用
- 存储系统性能关键的环境
- 需要考虑页面修改状态的系统

LFU/MFU算法

适用场景:

- 访问模式相对稳定的应用
- 需要长期统计信息的系统
- 特定的数据库和缓存系统

实验结果综合分析

性能排序

在大多数测试场景下的性能排序:

1. Enhanced CLOCK
2. LRU
3. CLOCK
4. LFU
5. FIFO
6. MFU

实现复杂度排序

从简单到复杂:

1. FIFO
2. CLOCK
3. Enhanced CLOCK
4. LFU/MFU
5. LRU

内存开销排序

从低到高:

1. FIFO
2. CLOCK
3. Enhanced CLOCK

4. LRU
5. LFU/MFU

Git 记录和上传

执行`git log`代码，得到结果如下图

```
kpmark@LAPTOP-AITJPLGC:/mnt/d/os-homework$ git log
commit 582f159b68fa01314ca0d19797f06ff21f250019 (HEAD -> main)
Author: kpmark <2585050765@qq.com>
Date:   Tue Jun 3 19:41:24 2025 +0800

    完成lab5代码

commit 0f0c18a9e42fdc91fe12372e6fb492b85748f1e4 (origin/main)
Author: kpmark <2585050765@qq.com>
Date:   Tue May 27 20:12:18 2025 +0800

    完成报告

commit e5ed2085ae6487282a30df99b4fe27e79ce57d97
Author: kpmark <2585050765@qq.com>
Date:   Tue May 27 19:26:53 2025 +0800
```

markzhang12345/os-homework: 大连理工大学操作系统课程作业